1.

体验 MySQL 在 InnoDB 存储引擎下的 MVCC 多版本并发控制，实现的事务 ACID 特性。请注意 Mysql 需要选用什么事务隔离级来支持 MVCC？请构造多用户多写多读案例来展现 MVCC 并发控制特性，解释各种结果产生的原因。

在MySQL中，首先创建一个数据库testdb

mysql -u root -p

create database testdb;
use testdb;

```
mysql> create database testdb;
Query OK, 1 row affected (0.00 sec)

mysql> use testdb;
Database changed
```

在testdb中创建一个新表，设置engine=InnoDB

```
create table test (
 id int(10) not null,
 name varchar(20) not null,
 flag int(5) not null,
 primary key(id)
) engine=InnoDB;
```

设置事务隔离等级为可重复读（repeatable read）

set session transaction isolation level repeatable read;

插入初始数据：

insert into test VALUES(1, 'LJP', 27);

insert into test VALUES(2, 'DJE', 42);

insert into test VALUES(3, 'OFW', 61);

insert into test VALUES(4, 'SLX', 15);

使用select * from test;查看当前表内的数据：

```
mysql> select * from test
    -> ;
+----+------+------+
| id | name | flag |
+----+------+------+
|  1 | LJP  |   27 |
|  2 | DJE  |   42 |
|  3 | OFW  |   61 |
|  4 | SLX  |   15 |
+----+------+------+
4 rows in set (0.00 sec)
```

开始一个事务：

start transaction;

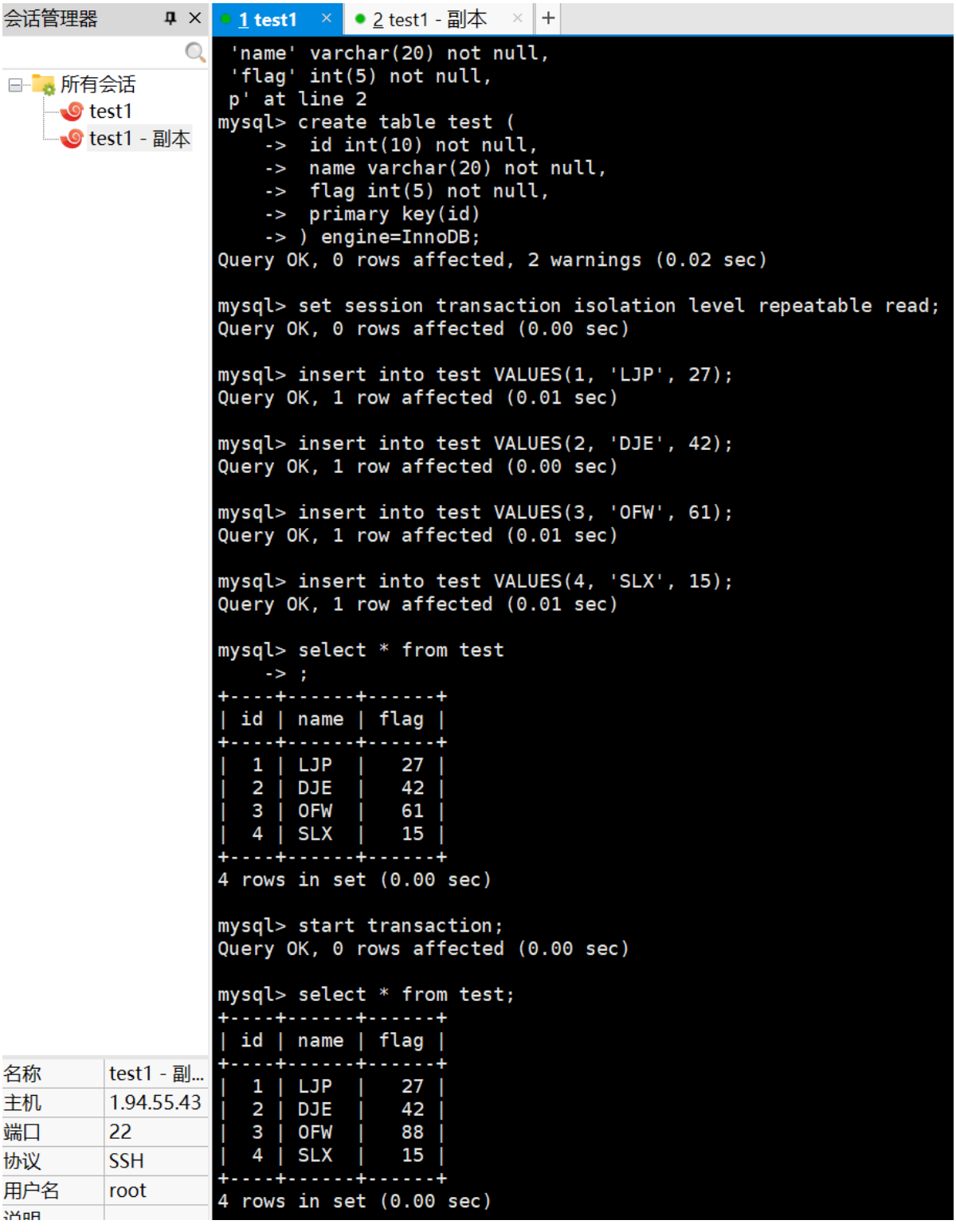
然后打开另一个会话，直接在xshell中完成即可。

更新id=3的flag为88

update test set flag=88 where id=3;

```
mysql> update test set flag=88 where id=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

在原先的会话中再次使用select * from test;查看当前表内的数据，可以发现已经成功更新：



再在第一个会话中更新（先不commit）：update test set flag=100 where id=4;

```
mysql> update test set flag=100 where id=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

到第二个会话中查询当前数据，发现此时无变化

```
mysql> select * from test;
+----+------+------+
| id | name | flag |
+----+------+------+
|  1 | LJP  |   27 |
|  2 | DJE  |   42 |
|  3 | OFW  |   88 |
|  4 | SLX  |   15 |
+----+------+------+
4 rows in set (0.00 sec)
```

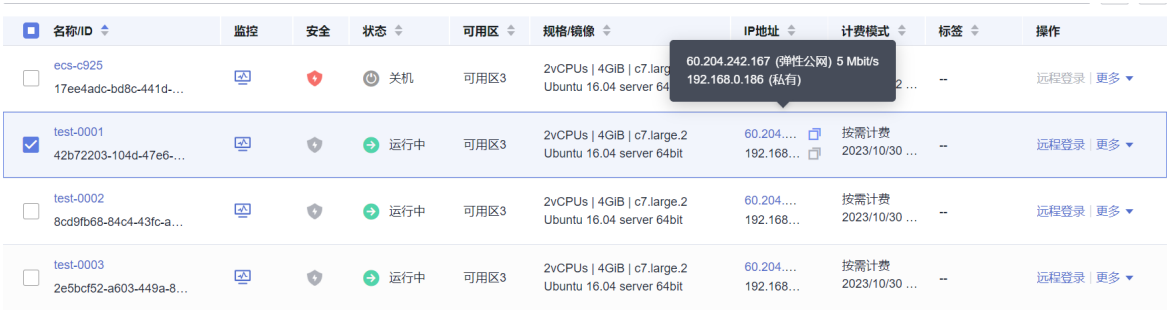
然后回到第一个终端commit

```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from test;
+----+------+------+
| id | name | flag |
+----+------+------+
|  1 | LJP  |   27 |
|  2 | DJE  |   42 |
|  3 | OFW  |   88 |
|  4 | SLX  |  100 |
+----+------+------+
4 rows in set (0.00 sec)
```

2.

体验 MongoDB 的 MVCC，数据集可自建或选用 yelp 数据集中的 test 集合中进行测试，测试方法同 MySQL。请对测试结果进行说明，并与 MySQL 的 MVCC 实验结果进行对比分析。建议创建 MongoDB 副本或分片集群，体验 MVCC 的不同效果（可选做其一）。



创建三台服务器(test-0001 test-0002 test-0003)，弹性公网ip分别为：

60.204.242.167

60.204.228.189

60.204.244.116

添加入方向和出方向规则，检验是否可以ping通：

```
Microsoft Windows [版本 10.0.22621.2428]
(c) Microsoft Corporation。保留所有权利。

C:\Users\xiaoy>ping 60.204.228.189

正在 Ping 60.204.228.189 具有 32 字节的数据:
来自 60.204.228.189 的回复: 字节=32 时间=65ms TTL=46
来自 60.204.228.189 的回复: 字节=32 时间=52ms TTL=46
来自 60.204.228.189 的回复: 字节=32 时间=52ms TTL=46
来自 60.204.228.189 的回复: 字节=32 时间=52ms TTL=46

60.204.228.189 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 52ms, 最长 = 65ms, 平均 = 55ms
```

三台服务器分别创建数据存放的目录：

```
root@test-0001:~# su - mongodb
No directory, logging in with HOME=/
root@test-0001:~# mkdir /usr/local/mongodb
root@test-0001:~# cd /usr/local/mongodb
root@test-0001:/usr/local/mongodb# mkdir -p data/shard11
root@test-0001:/usr/local/mongodb# mkdir -p data/shard21
root@test-0001:/usr/local/mongodb# mkdir -p data/config
root@test-0001:/usr/local/mongodb# touch data/shard11.log
root@test-0001:/usr/local/mongodb# touch data/shard21.log
root@test-0001:/usr/local/mongodb#
```

```
root@test-0002:~# su - mongodb
No directory, logging in with HOME=/
root@test-0002:~# mkdir /usr/local/mongodb
root@test-0002:~# cd /usr/local/mongodb
root@test-0002:/usr/local/mongodb# mkdir -p data/shard12
root@test-0002:/usr/local/mongodb# mkdir -p data/shard22
root@test-0002:/usr/local/mongodb# mkdir -p data/config
root@test-0002:/usr/local/mongodb# touch data/shard12.log
root@test-0002:/usr/local/mongodb# touch data/shard22.log
root@test-0002:/usr/local/mongodb#
```

```
root@test-0003:~# su - mongodb
No directory, logging in with HOME=/
root@test-0003:~# mkdir /usr/local/mongodb
root@test-0003:~# cd /usr/local/mongodb
root@test-0003:/usr/local/mongodb# mkdir -p data/shard13
root@test-0003:/usr/local/mongodb# mkdir -p data/shard23
root@test-0003:/usr/local/mongodb# mkdir -p data/config
root@test-0003:/usr/local/mongodb# touch data/shard13.log
root@test-0003:/usr/local/mongodb# touch data/shard23.log
root@test-0003:/usr/local/mongodb#
```

分别完成shard1和shard2的replica set配置：

```
root@test-0001:/usr/local/mongodb# mongod --shardsvr --replSet shard1 --port 27017 --dbpath /usr/loc
al/mongodb/data/shard11 --oplogSize 2048 --logpath /usr/local/mongodb/data/shard11.log --logappend -
-bind_ip=0.0.0.0 --fork
about to fork child process, waiting until server is ready for connections.
forked process: 8502
child process started successfully, parent exiting
```

```
root@test-0001:/usr/local/mongodb# mongod --shardsvr --replSet shard2 --port 27018 --dbpath /usr/local/mongodb/data/shard21 --oplogSize 2048 --logpath /usr/local/mongodb/data/shard21.
log --logappend --bind_ip=0.0.0.0 --fork
about to fork child process, waiting until server is ready for connections.
forked process: 8612
child process started successfully, parent exiting
```

初始化shard1和shard2的replica set：

shard1，在某一台服务器上执行：

mongo 60.204.242.167:27017

config = { *id: 'shard1', members: [{*id: 0, host: '60.204.242.167:27017'}, {*id: 1, host: '60.204.228.189:27017'}, {*id: 2, host: '60.204.244.116:27017'}] }

rs.initiate(config);

shard2，在某一台服务器上执行：

mongo 60.204.242.167:27018

config = { *id: 'shard2', members: [{*id: 0, host: '60.204.242.167:27018'}, {*id: 1, host: '60.204.228.189:27018'}, {*id: 2, host: '60.204.244.116:27018'}] }

rs.initiate(config);

```
root@test-0001:/usr/local/mongodb# mongo 60.204.242.167:27017
MongoDB shell version v4.4.17
connecting to: mongodb://60.204.242.167:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2498494b-f982-4229-98f7-fcfaed2591d6") }
MongoDB server version: 4.4.17
---
The server generated these startup warnings when booting:
        2023-10-30T20:31:52.990+08:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
        2023-10-30T20:31:53.632+08:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2023-10-30T20:31:53.632+08:00: You are running this process as the root user, which is not recommended
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> config = { _id: 'shard1', members: [ {_id: 0, host: '60.204.242.167:27017'}, {_id: 1, host: '60.204.228.189:27017'}, {_id: 2, host: '60.204.244.116:27017'}] }
{
        "_id" : "shard1",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "60.204.242.167:27017"
                },
                {
                        "_id" : 1,
                        "host" : "60.204.228.189:27017"
                },
                {
                        "_id" : 2,
                        "host" : "60.204.244.116:27017"
                }
        ]
}
> rs.initiate(config);
{ "ok" : 1 }
shard1:SECONDARY>
```

```
> config = { _id: 'shard2', members: [ {_id: 0, host: '60.204.242.167:27018'}, {_id: 1, host: '60.204.228.189:27018'}, {_id: 2, host: '60.204.244.116:27018'}] }
{
        "_id" : "shard2",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "60.204.242.167:27018"
                },
                {
                        "_id" : 1,
                        "host" : "60.204.228.189:27018"
                },
                {
                        "_id" : 2,
                        "host" : "60.204.244.116:27018"
                }
        ]
}
> rs.initiate(config);
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698669536, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1698669536, 1)
}
shard2:SECONDARY>
```

配置config server：

在三台服务器上分别执行：

mongod --configsvr --replSet config --dbpath /usr/local/mongodb/data/config --port 20000 --logpath /usr/local/mongodb/data/config.log --logappend --bind_ip=0.0.0.0 --fork

在某一台服务器上执行：

mongo 60.204.242.167:20000

config = { *id: 'config', members: [{*id: 0, host: '60.204.242.167:20000'}, {*id: 1, host: '60.204.228.189:20000'}, {*id: 2, host: '60.204.244.116:20000'}] }

rs.initiate(config);

```
> config = { _id: 'config', members: [ {_id: 0, host: '60.204.242.167:20000'}, {_id: 1, host: '60.204.228.189:20000'}, {_id: 2, host: '60.204.244.116:20000'}] }
{
        "_id" : "config",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "60.204.242.167:20000"
                },
                {
                        "_id" : 1,
                        "host" : "60.204.228.189:20000"
                },
                {
                        "_id" : 2,
                        "host" : "60.204.244.116:20000"
                }
        ]
}
> rs.initiate(config);
{
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1698669771, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698669771, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1698669771, 1)
}
config:SECONDARY>
```

配置mongos：

在三台服务器上分别执行：

mongos --configdb config/60.204.242.167:20000,60.204.228.189:20000,60.204.244.116:20000 --port 30000 --logpath /usr/local/mongodb/data/mongos.log --logappend --bind_ip=0.0.0.0 --fork

使用mongos：

mongo 60.204.242.167:30000

切换到admin，添加分片：

use admin;

db.runCommand({addshard:"shard1/60.204.242.167:27017,60.204.228.189:27017,60.204.244.116:27017",name:"s1", maxsize:20480});

db.runCommand({addshard:"shard2/60.204.242.167:27018,60.204.228.189:27018,60.204.244.116:27018",name:"s2", maxsize:20480});

```
mongos> use admin;
switched to db admin
mongos> db.runCommand({addshard:"shard1/60.204.228.189:27017,60.204.242.167:27017, 60.204.244.116:27017",name:"s1", maxsize:20480});
{
        "ok" : 0,
        "errmsg" : "in seed list shard1/60.204.228.189:27017,60.204.242.167:27017, 60.204.244.116:27017, host  60.204.244.116:27017 does not belong to replica set shard1; found { topologyVersion: { processId: ObjectId('653f7c3cd1e8e41120d20c6e'), counter: 6 }, hosts: [ \"60.204.228.189:27017\", \"60.204.242.167:27017\", \"60.204.244.116:27017\" ], setName: \"shard1\", setVersion: 1, ismaster: true, secondary: false, primary: \"60.204.228.189:27017\", me: \"60.204.228.189:27017\", electionId: ObjectId('7fffffff0000000000000001'), lastWrite: { opTime: { ts: Timestamp(1698663825, 1), t: 1 }, lastWriteDate: new Date(1698663825000), majorityOpTime: { ts: Timestamp(1698663825, 1), t: 1 }, majorityWriteDate: new Date(1698663825000) }, maxBsonObjectSize: 16777216, maxMessageSizeBytes: 48000000, maxWriteBatchSize: 100000, localTime: new Date(1698663828408), logicalSessionTimeoutMinutes: 30, connectionId: 24, minWireVersion: 0, maxWireVersion: 9, readOnly: false, compression: [ \"snappy\", \"zstd\", \"zlib\" ], ok: 1.0, $clusterTime: { clusterTime: Timestamp(1698663825, 1), signature: { hash: BinData(0, 0000000000000000000000000000000000000000), keyId: 0 } }, operationTime: Timestamp(1698663825, 1) }",
        "code" : 96,
        "codeName" : "OperationFailed",
        "operationTime" : Timestamp(1698663827, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698663827, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> db.runCommand({addshard:"shard2/60.204.228.189:27018,60.204.242.167:27018, 60.204.244.116:27018",name:"s2", maxsize:20480});
{
        "ok" : 0,
        "errmsg" : "in seed list shard2/60.204.228.189:27018,60.204.242.167:27018, 60.204.244.116:27018, host  60.204.244.116:27018 does not belong to replica set shard2; found { topologyVersion: { processId: ObjectId('653f88d203af49fdeea2c389'), counter: 6 }, hosts: [ \"60.204.228.189:27018\", \"60.204.242.167:27018\", \"60.204.244.116:27018\" ], setName: \"shard2\", setVersion: 1, ismaster: true, secondary: false, primary: \"60.204.228.189:27018\", me: \"60.204.228.189:27018\", electionId: ObjectId('7fffffff0000000000000001'), lastWrite: { opTime: { ts: Timestamp(1698663830, 1), t: 1 }, lastWriteDate: new Date(1698663830000), majorityOpTime: { ts: Timestamp(1698663830, 1), t: 1 }, majorityWriteDate: new Date(1698663830000) }, maxBsonObjectSize: 16777216, maxMessageSizeBytes: 48000000, maxWriteBatchSize: 100000, localTime: new Date(1698663838847), logicalSessionTimeoutMinutes: 30, connectionId: 23, minWireVersion: 0, maxWireVersion: 9, readOnly: false, compression: [ \"snappy\", \"zstd\", \"zlib\" ], ok: 1.0, $clusterTime: { clusterTime: Timestamp(1698663830, 1), signature: { hash: BinData(0, 0000000000000000000000000000000000000000), keyId: 0 } }, operationTime: Timestamp(1698663830, 1) }",
        "code" : 96,
        "codeName" : "OperationFailed",
        "operationTime" : Timestamp(1698663838, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698663839, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

（上图为报错，因为addshard字段内严格不允许出现空格）

```
root@test-0001:/usr/local/mongodb# mongo 60.204.242.167:30000
MongoDB shell version v4.4.17
connecting to: mongodb://60.204.242.167:30000/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d7addd2e-c143-4e61-87f5-45ff525820b8") }
MongoDB server version: 4.4.17
---
The server generated these startup warnings when booting:
        2023-10-30T20:44:18.813+08:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2023-10-30T20:44:18.813+08:00: You are running this process as the root user, which is not recommended
---
mongos> use admin;
switched to db admin
mongos> db.runCommand({addshard:"shard1/60.204.242.167:27017,60.204.228.189:27017,60.204.244.116:27017",name:"s1", maxsize:20480});
{
        "shardAdded" : "s1",
        "ok" : 1,
        "operationTime" : Timestamp(1698669951, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698669951, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> db.runCommand({addshard:"shard2/60.204.242.167:27018,60.204.228.189:27018,60.204.244.116:27018",name:"s2", maxsize:20480});
{
        "shardAdded" : "s2",
        "ok" : 1,
        "operationTime" : Timestamp(1698669965, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698669965, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

激活数据库分片：

创建一个名为testdb的数据库：

use testdb

```
mongos> use testdb
switched to db testdb
```

激活分片：

sh.enableSharding("testdb")

```
mongos> sh.enableSharding("testdb")
{
        "ok" : 1,
        "operationTime" : Timestamp(1698670042, 8),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698670042, 8),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

使用sh.status()查看数据库当前情况：

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("653fa4d673838d5e44d4bc05")
  }
  shards:
        {  "_id" : "s1",  "host" : "shard1/60.204.228.189:27017,60.204.242.167:27017,60.204.244.116:27017",  "state" : 1 }
        {  "_id" : "s2",  "host" : "shard2/60.204.228.189:27018,60.204.242.167:27018,60.204.244.116:27018",  "state" : 1 }
  active mongoses:
        "4.4.17" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                28 : Success
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                s1      996
                                s2      28
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "testdb",  "primary" : "s2",  "partitioned" : true,  "version" : {  "uuid" : UUID("1b3b503d-a2b2-4e9c-b435-be52f5d5c4dc"),  "lastMod" : 1 } }
mongos>
```

创建一个新的集合testc：

db.createCollection("testc")

```
mongos> db.createCollection("testc")
{
        "ok" : 1,
        "operationTime" : Timestamp(1698673262, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1698673262, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

插入一些数据

db.testc.insertMany([
 {"*id*": 1, "name": "LJP", "age": 27},
 {"id": 2, "name": "DJE", "age": 42},
 {"*id*": 3, "name": "OFW", "age": 61},
 {"id": 4, "name": "SLX", "age": 15}
]);

```
mongos> db.testc.insertMany([
...     {"_id": 1, "name": "LJP", "age": 27},
...     {"_id": 2, "name": "DJE", "age": 42},
...     {"_id": 3, "name": "OFW", "age": 61},
...     {"_id": 4, "name": "SLX", "age": 15}
... ]);
{ "acknowledged" : true, "insertedIds" : [ 1, 2, 3, 4 ] }
mongos>
```

通过db.testc.find()可以查看当前集合中的数据：

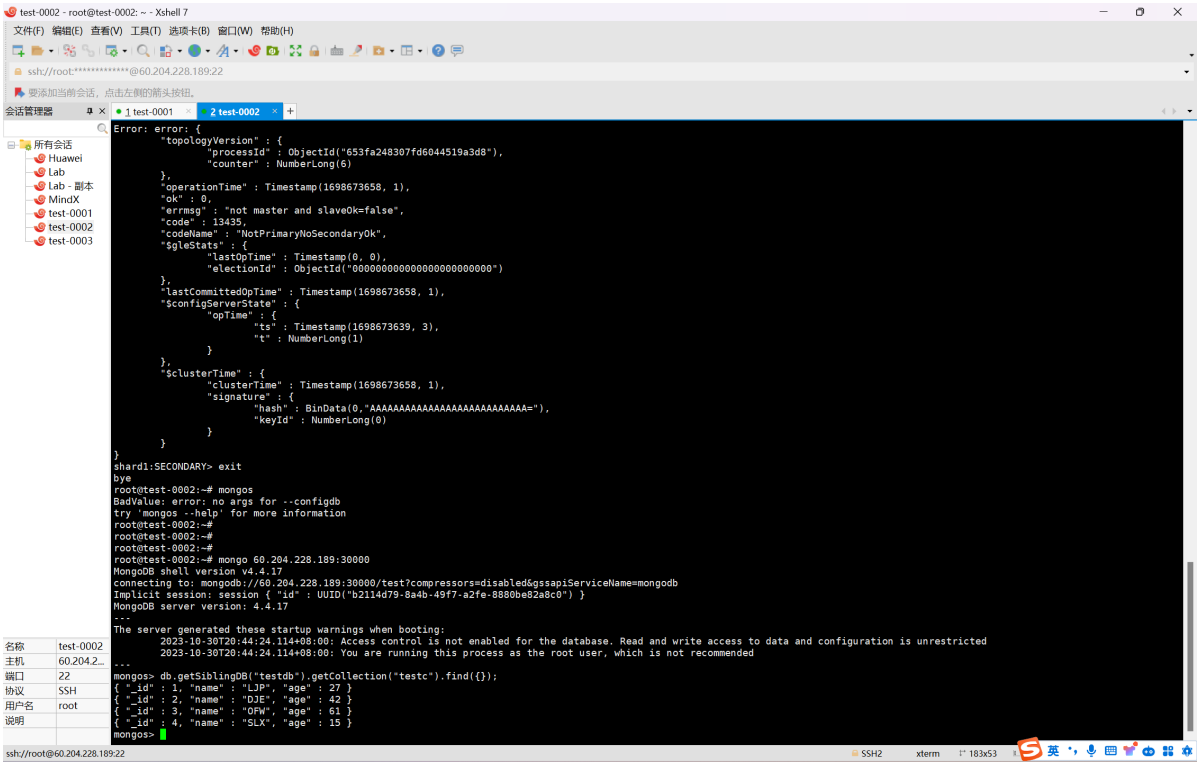
```
mongos> db.testc.find()
{ "_id" : 1, "name" : "LJP", "age" : 27 }
{ "_id" : 2, "name" : "DJE", "age" : 42 }
{ "_id" : 3, "name" : "OFW", "age" : 61 }
{ "_id" : 4, "name" : "SLX", "age" : 15 }
```

此时打开test-0002服务器进入mongos：

mongo 60.204.228.189:30000

查询数据：

db.getSiblingDB("testdb").getCollection("testc").find({});

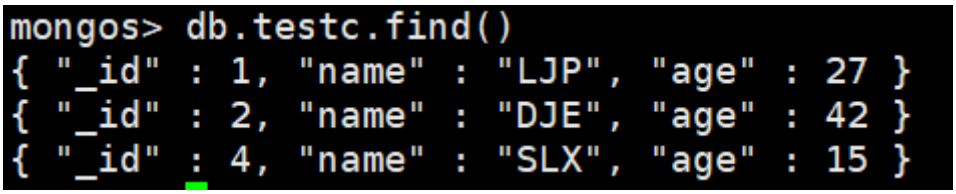


删除一条数据：

db.getSiblingDB("testdb").getCollection("testc").deleteOne({ _id: 3 })

```
mongos> db.getSiblingDB("testdb").getCollection("testc").deleteOne({ _id:3 })
{ "acknowledged" : true, "deletedCount" : 1 }
```

再回到test-0001服务器，通过db.testc.find()查看当前集合中的数据：



MySQL和mongoDB的MVCC对比分析：

MySQL和MongoDB都实现了MVCC（多版本并发控制）机制，用以解决读写冲突的并发控制。在MVCC机制中，为事务分配单向增长的时间戳，并为每个修改保存一个版本，版本与事务时间戳关联。读操作只会读取该事务开始前的数据库快照，从而避免阻塞其他读操作。

虽然二者都采用了MVCC，但在具体实现上存在显著差异。MySQL的MVCC是通过保存数据的历史版本来实现的，每个数据项都有一个时间戳timestamp，可以追踪到创建和修改的时间点。这种实现方式使得MySQL能够提供严格的可重复读，保证读取操作不会看到未提交（commit）的修改。

MongoDB的MVCC实现更复杂。在MongoDB中，每个文档都有一个包含多个版本的时间戳数组，也可以看作一个topologyVersion（拓扑版本号）。当文档被修改时，旧版本并不会被删除，而是存储在数组中。这种方式允许MongoDB在读取数据时查看过时的数据版本，从而实现非阻塞读操作。